SUNFLOWER: A Trust Execution Environment on RISCV Based on Tagged Memory

Xu Jinyan 316010**** Duan Yuxuan 316010**** Lin Yizhu 316010****

***Abstract*—This is a report for the class: Network Security Theory and Practice, our project is aimed to build a execution environment in RISC-V using tagged memory technology. RISC-V is an open and extensible instruction set architecture designed by UC Berkly, because of its low overhead, it's very suitable for using in the embedded devices and internet of things. As with the various system vulnerabilities recently exposed, operating systems are no longer fully trusted. More and more people choose to using the hardware security method, like Trust Zone(ARM), SGX(Intel). However, there are still no officically supported safety standards for RISC-V.**

Here, we design a tagged memory hardware security architecture, inspired by the paper[1]. Tagged memory has been proposed as a fine-grained security mechanism to support protection of data flow, pointers or capabilities for a long time, but none of the existing schemes provide efficient and flexible at the same time, it is still a open problem. We using tag-aware instruction and a simple kernel to achieve a fine-grained isolation. We also provide a series of tools for easy to use, a gcc and objdump supported tag-aware instructions. And we finish our logical design in Spike, a RISC-V simulator.

***Index Terms*—RISC-V, tagged memory, tag-aware instruction set architecture.**

I. Introduction

IoT devices are rapidly entering everyday life. However, these devices have a great risk to be attacked since the code of these devices are complex thus potential to have bugs. To ensure the security of the applications running on these devices even when the devices are compromised, a kind of skill called Trusted Execution Environment(TEE) can be of great help. The main idea of TEE is to use isolated execution to protect trusted applications from compromised operating systems or other untrusted software on the same device.

Recent years have witnessed many implementations of TEE on different systems. ARM TrustZone and Intel SGX are two widely applied TEE architectures. ARM TrustZone divides the computer resources into a secure and non-secure world on hardware level.[1] The secure world can access all system resources while the non-secure world can only access data in the non-secure world or call secure applications through designed secure entry points. With TrustZone architecture, if sensitive data and code are hidden in the secure world, they cannot be accessed by untrusted applications or system. However, TrustZone needs a secure kernel to manage the secure world, providing secure system services, thus enlarge the trusted computing base(TCB), which increases the risk. Intel SGX uses an isolated container called enclave which hold sensitive part of applications.[2] Unlike TrustZone, SGX does not need any privileged supervisor like a trusted kernel, thus reduces TCB. But SGX needs complex instructions to manage enclaves.

Resource-constrained devices like IoT devices often have little memory, thus needs fine-grained memory management. The isolation boundary the existing implementations of TEE provides are not fine-grained enough for small IoT devices to apply.[3] Inspired by the work in [3], we use a technique called *tagged memory* to provide flexible isolation for low-end devices. The main method of tagged memory technique is to associate memory blocks with additional metadata called tag, used for access control and other memory management. There have been active new research on tagged memory, such as use this technique help data flow tracking.[4]. However current tagged memory scheme are not efficient enough on small IoT devices[3], which is the aim of our work.

This work is based on the instruction set architecture RISC-V.[5] RISC-V is suitable for development on IoT devices due to its simple instruction architecture. Also it is open and support instruction extension. On the other hand, the scheme which uses tagged memory technique to implement TEE hasn't been supported on RISC-V. These are the reasons why we choose to implement our work on RISC-V. RISC-V defines three different privilege mode: machine-mode(M), supervisor-mode(S) and user-mode(U). M-mode can access all the system resources and are meant for emulating the machine, while S-mode and U-mode are respectively used by operate systems and user applications.

In this work, we present SUNFLOWER, a RISC-V based TEE implementation with tagged memory applied. This is a hardware-software co-design work. In the hardware level, we add instructions that can load and store data with tag check to the RISC-V simulator spike. In the software level, we provide a secure interface for applications to call and set part of their code or data with trusted tag. We associate each memory word with 2 bits tag, thus achieve fine-grained isolation with low overhead. We estimate our work on a simple encryption program and prove it can work as expected.

This report will introduce our design frame and details of hardware and software design, followed by evaluation and analysis of our work.

II. SUNFLOWER Design Frame

This work is a hardware and software co-design. The hardware design is based on a RISC-V simulator called spike, while the software part is designed on Amazon FreeRTOS embedded system. To support the tagged memory in our work, a tag-aware instruction set is designed as an extension to the original RV64ISA of RISC-V. This instruction set includes instructions to visit memory unit with tags, helping our system realize access control to memory with different tags. The instruction decoder and MMU are also modified according to the structure of this newly designed instruction set. On the system layer, we design and implement a set of system services, which are run in high privilege mode and perform services like tag update.

III. Design Details

In this section, we will introduce our hardware-software co-design system in detail. Hardware design includes different tags and tag-aware instruction sets, as well as the modification on the datapath and decoder of CPU.

A. Hardware Design

To implement tagged memory aided trusted execution environment, the basic part is to modify the hardware so that it can support tagged memory. We design a 2-bit tag system and a tag-aware instruction set to visit memory with tags. Based on the new instructions, we extend the original CPU to support them.

1) Tag System Design: Memory tags in SUNFLOWER system each has 2 bits, resulting in 4 different types of tags, which are N-tag, TU-tag, TS-tag and TC-tag. N-tag is for memory in normal world. TU-tag is for code and data in user mode trusted enclaves, while TS-tag is in supervisor mode trusted enclaves. TC-tag is for entry point from normal state to trusted state. It is designed just for instructions, not for data, which is different from the others.

2) Tag-aware Instructions: We extend the original RISC-V instruction architecture with a set of tag-aware instructions to support tagged memory operations. The newly designed instructions are based on I/S type instructions and can perform access control for load and store operations according to the destination memory tag.

The original load instructions are extended to be load check tag instructions(LCT). LCT instructions can compare the tag of the destination memory unit with the expected tag specified in the instruction, check if this load request is legal, and raise exception if it abuses the tag isolation policy. The high 12 bits of the original I-type instructions is immediate operator, used as address offset. We use the highest 2 bits to store the expected tag.

Similarly, the original store instructions are extended to be store check tag instructions(SCT). SCT instructions work the same as LCT instructions, store to destination memory unit if the memory tag and the expected tag match and raise exception if they don't. The immediate fields in S-type instructions consists of the high 7 bits and 8-12 4bits, used for offset in addressing. The highest 2 bits are used to store tags.

This design that takes 2 bits from the original immediate fields will affect the addressing space size. The original addressing space is from -4096 to 4096

B. Basic Execution Environment Theoretical Concept

Since RISC-V program could not run by itself, applications ,based on RISC-V, need kernel support to work. For instance, the RISC-V Proxy Kernel, pk, is a lightweight application execution environment that can host statically-linked RISC-V ELF binaries. It is designed to support tethered RISC-V implementations with limited I/O capability and thus handles I/O-related system called by proxying them to a host computer. Tethered System, as its name, refers to those system that cannot stand alone, which means that they may not have complete facilities and depend on a host system to boot. Speaking of boot, pk goes with the Berkely Boot Loader, denoted as bbl. It is the supervisor part of this execution environment and need to handle basic booting work, like initialization.

To be more detailed, RISC-V Proxy Kernel implement a communication between RISC-V Target running proxy kernel and x86 Host running frontend server. The bridge between them is called Host-Target Interface (HTIF), which corresponds to riscv-fesvr project.

At first, many experimentations related our final goal were done by using pk. Then, based on these theoretical concepts learned from pk, our project determines to build applications on FreeRTOS, which is a open source real-time operating system kernel for embedded devices. What is more, it has been ported to RISC-V micro-controller platform.

C. Custom Functions Supported by FreeRTOS Port

As said before, FreeRTOS could support RISC-V perfectly and detailed implementation is contained in its portable files, to be more specifically, mostly in port.c.

In this port, it is most significant that we rewrite and complete its syscall trap to support our design. What is noteworthy is that trap in FreeRTOS is generalized trap, which simulates both hardware and software interruption. In this trap, we need to handle a mode-change request, which transfer current mode to machine mode. This design is on the purpose of enclave service management, which need to be dealt with in machine mode. Detailed process is written in assembly language by modifying mstatus register to change current mode.

D. Enclave Implementation Details

Process in FreeRTOS is called task, which is quite important concept of this operating system. As we all know, in Linux

we sometimes call it task, too, like task_struct. Just like task_struct, every task in FreeRTOS need a xTaskParameter to initialize, which include information about detailed function to run, its name, stack allocation and so on. Besides this, every task, seen as an individual application, also need another parameter, denoted as xEnclaveRegions, to mark where belongs to enclave, and an function address list, denoted as xTaskEntryPoints, to mark where secure functions begin.

After introducing our resources to be prepared, it is time to figure out our basic idea of enclave implementation. Our aim is to put all memory, which need to be tagged and belongs to these task, together during linking stage. Then if our memory is continuous, it would be much convenient to set tag and do other management. To achieve this purpose, divide memory four part, normal data, normal function, secure data and secure function and denote them using different marks, like NORMAL_DATA, NORMAL_FUNCTION, SECURE_DATA and SECURE_FUNCTION. These marks correspond to their __attribute__, consist of name, mode, type and position, which could determine a section in memory. Positions are set as ATTR_MIDDLE firstly and will be explained later.

Position is the most ingenious trick in the method. Create four pointers to mark where normal and secure regions start and end. They have not been initialized, as they are just symbols without any exact meaning now. Then set their positions to ATTR_START and ATTR_END. So far, all functions and data are sectioned into chucks of memory. Hence, in linking stage, pack text and data up, and sort them by using position. In order to do that, we define ATTR_START as 'a', which is the beginning of alphabet, ATTR_END as 'z', the ending, and ATTR_MIDDLE 'b' or other letter in the middle pf 'a' and 'z'. Then the sort could be just implement by linker script using SORT_BY_NAME.

Finally, get each part's address at load-time because linker cannot know symbol at link time. In the other words, we do this in task initialization. Until now, we could set enclave and tag uniformly by using our encapsulated API, which would be described in more detailed later.

E. Enclave Service API Encapsulation

Our trusted supervisor provides FreeRTOS a series of API to manipulate enclaves. Every enclave has a structure, denoted as TTCB, to keep information, like region address, magic number for security and so on. In these operating system services, we use mutex lock to maintain mutual exclusiveness of TTCB and tag demanding memory by using checked load and store instructions, which also are encapsulated.

IV. Demo Design

We design a demo program to show our design works. Boot loader runs in machine mode, first load our tag service, then is the FreeRTOS kernel. When everything is ready, switch to user mode to run our test program. Part of the memory is allocated for the trusted environment, then TU is marked for this part, and TC is marked for the first instruction as the entry. Then the program and relevant data are loaded, and the trusted program can be called after initialization. When program finished, we will release this part and re-mark the N mark.[images]

The operation of the trust encryption function is to use the tags marked with TU to encrypt the data marked with N, that is to say, use the trust data to encrypt the untrusted data. In the demo, test_n corresponds to the untrusted data, and test_s corresponds to the trust data.We encrypt data in bytes. When trust function is completed, we return to untrust world and print our encrypted result.[image]

At the same time, we designed an untrusted program to try to access the trusted world to prove our design works. As shown in the figure, this program tried to directly output the key, but it failed, which caused a trap to terminate this process. The following output information is the register content at this time.[image]

V. MMIC designs

The MMICs were fabricated in a $0.15\mu m$ gate length process. The PAs have been designed for Envelope Tracking application. The output networks are optimized on efficiency performances. Fig. 1 shows the X-Band power amplifiers charaterized in both amplifier and rectifier modes in this paper:

- Circuit-A (Fig. 1a) is a $10 \times 100\mu m$ single transistor amplifier ;
- Circuit-B (Fig. 1b) is a single stage amplifier that combines two $10 \times 100\mu m$ transistors with a reactive combiner.

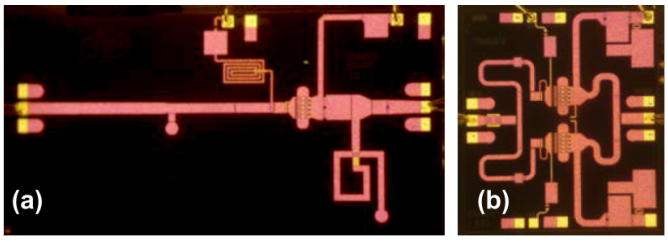

Fig. 1. X-Band MMIC power amplifiers used as rectifiers : (a) Circuit-A: Single stage, $10 \times 100\mu m$. $3.8 \times 2.3mm^2$ and (b) Circuit-B: Single stage, two $10 \times 100\mu m$. $2.0 \times 2.3mm^2$.

VI. Measurement Setup

The measurement setup is based on the SWAP X-402 [**?**], [**?**]. This 4 channels time-domain receiver, working like a LSNA [**?**] thanks to bi-directional couplers, acquires absolutes incident and reflected waves at DUT's inputs and output ports. Thanks to a relative SOLT and a power calibration, the setup provides time domain RF waveforms for voltages and currents at the calibration reference planes. The SWAP enables measurements in a 30 GHz RF bandwidth : only the fundamental and the second harmonic were measured.

Nevertheless, this paper focus only on the fundamental RF frequency.

During power amplifier measurements, the RF signal is injected at the input of the DUT (gate's port). The DC-to-RF efficiency is calculated according the the output power, the input power and the bias consumption in term of Power Added Efficiency (η_{PA}) or Drain Efficiency (η_{DE}) such as :

$$\eta_{PA} = \frac{P_{out}(f_0) - P_{in}(f_0)}{P_{DC}} \; ; \; \eta_{DE} = \frac{P_{out}(f_0)}{P_{DC}} \tag{1}$$

Power amplifier characterization where performed with a $50\,\Omega$ RF load applied to the drain RF port.

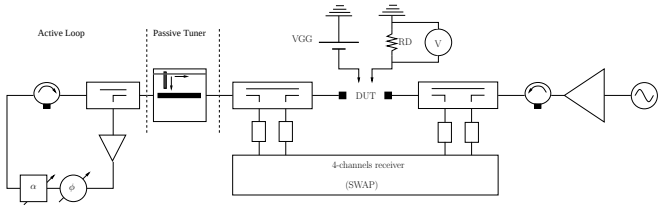

Fig. 2. Time-domain measurement setup for rectifier characterizations. A load-pull is applied on the gate RF port of the DUT. It as been perform with a passive tuner and an active loop for highly reflective load impedance at 10.1 GHz.

Regarding the rectifier characterizations, the RF signal is injected at the RF drain port of the DUT. The output of the rectifier is the DC path of the drain port. The PA is only biased on the gate. In order to obtain the best efficiency, DC-load on the drain (RD) and RF load on the gate are tuned as depicted on Fig. 2. The load-pull is performed at the fundamental frequency (10.1 GHz) with a passive tuner. High magnitude on reflexion coefficients have been reached with an active loop. The RF-to-DC or rectifier efficiency is given by :

$$\eta_R = \frac{P_{DC}}{P_{RF\,injected}} = \frac{2|V_{DC}|^2}{RD \times \Re\{V_{drain}(f_0)\, I^*_{drain}(f_0)\}} \tag{2}$$

Power amplifiers have been measured in a coaxial test-fixture. The error-boxes of the test-fixture have been extracted thanks to a TRL calkit. Performances given is this paper are de-embedded to the input and output GSG ports pictured on Fig. 1.

VII. Measured Results

A. Circuit-A: one $10 \times 100\mu m$

Circuit-A, working as an amplifier, loaded with $50\,\Omega$ and measured at $f_0 = 10.1\,GHz$, exibits, as shown in Fig. 3, its best efficiency $\eta_{PA} = 67.87\%$ and $\eta_{DE} = 78.36\%$ with $P_{in} = 26.42\,dBm$ and $P_{out} = 35.16\,dBm$. The best bias point for efficiency is VGG=-4.0V and VDD=20V. The design is optimized for a bias point close to class-B.

This circuit has been characterized as a rectifier with the bench depicted on Fig. 2. In order to optimize the RF-to-DC efficiency (η_R), we can adjust 3 parameters : the gate bias voltage (VGG), the DC load impedance at the output

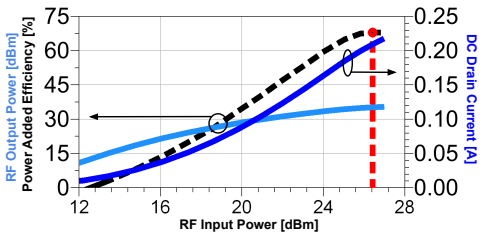

Fig. 3. Circuit-A best efficiency performances as an amplifier are measured with $VGG = -4.0\,V$ and $VDD = 20\,V$. Light blue curve is the output power, dark blue is the DC current on the drain and the efficiency (η_{PA}) is traced with a black dashed line. The maximum efficiency is $\eta_{PA} = 67.87\%$ at $P_{in} = 26.42\,dBm$ (red dashed line).

(RD) and the RF load impedance on the gate (Z_{load} or Γ_{load}).

- Z_{load} is a prime of importance regarding the rectifying efficiency of the circuit as illustrated for S-band in [**?**].
- VGG should correspond to a class-B or class-C to be compliant with the rectifier effect on a transistor. VGG has a moderate influence on the efficiency compared to Z_{load}. In this work, we noticed a deep class-C bias point make possible to reach the best rectifying efficiency. Fig. 4 illustrates a VGG sweep for fixed Z_{load} and RD. $\eta_R = 60.37\%$ has been measured at $P_{RF} = 34.63\,dBm$ but Z_{load} is not optimized (limited by the passive tuner during our automated multi-sweep measurements).

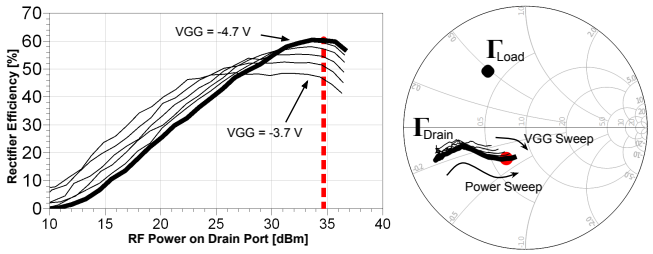

Fig. 4. Circuit-A rectifying performances for several VGG (from $-3.7\,V$ to $-4.7\,V$ with a $0.2\,V$ step). The gate's RF port is loaded at fundamental frequency, with $Z_{load} = (17.9 + j\,24.0)\Omega$ and the drain DC load impedance is $RD = 100\Omega$.

- RD range should be limited by the DC voltage and current values the transistor can handle. This value will establish the DC output power distribution between voltage and current. This parameters impacts the RF impedance at the input of the rectifier (drain's RF port) but has a limited influence on the maximum measured efficiency.

Influences of those 3 parameters are summerized in table I.

After optimization of Z_{load}, RD and VGG on circuit-A, the best rectifying efficiency $\eta_R = 64.40\%$ is obtained at $VGG = -4.7\,V$, $RD = 100\Omega$ and $Z_{load} = (8.45 + j\,24.5)\Omega$. The RF power sweep results are depicted on Fig. 5. Z_{load} has been applied thanks to the active loop. The RF power

measured at the gate's port reference plane is displayed too and demonstrates the fact that the PA can work as an efficient self-synchroneous rectifier thanks to a passive load applied to the gate's RF port.

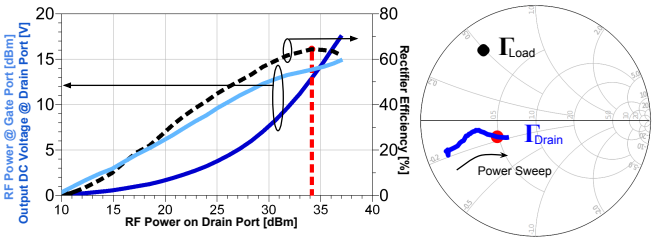

Fig. 5. Circuit-A : Rectifier efficiency η_R (black dashed line), DC output drain voltage (dark blue) and RF power at the gate reference plane (light blue) versus Injected drain RF power. The best rectifying performance is $\eta_R = 64.4\%$ at $P_{RF} = 34.14\,dBm$ (red dashed line). The smith chart on the right shows the optmal load impedance presented at the gate port (black dot) and the drain impedance of the DUT (blue curve) during the power sweep.

TABLE I
IMPACT ON RECTIFIER EFFICIENCY AND DRAIN RF IMPEDANCE

	Efficiency	RF drain impedance
Z_{load}	High	Medium
RDD	Low	High
VGG	Medium	Low

B. Circuit-B: two $10 \times 100\mu m$

Circuit-B has been characterized in the same manner than circuit-A for both amplifier and rectifier modes. Regarding the amplifier measurements, with a $50\,\Omega$ load, at $10.1\,GHz$, circuit B exhibits the best efficiency $\eta_{PA} = 56.47\%$ and $\eta_{DE} = 65.75\%$ with $P_{in} = 26\,dBm$ and $P_{out} = 35\,dBm$. The bias point is VGG=-3.4V and VDD=20V. The design of the amplifier is optimized for deep class-AB.

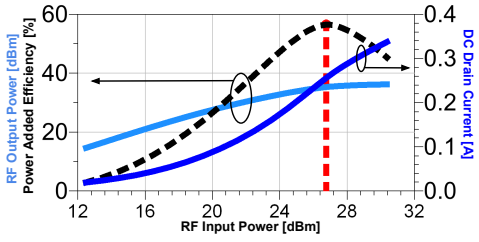

Fig. 6. Circuit-B as an amplifier: PAE η_{PA} (black dashed line), Output RF power (light blue) and DC drain current (dark blue) versus input RF power. The best efficiency is $\eta_{PA} = 56.47\%$ at $P_{in} = 26\,dBm$ (red dashed line) with $VGG = -3.4\,V$, $VDD = 20\,V$.

Rectifier measurements were performed on circuit-B. After an optimization on the variables Z_{load}, RD and VGG. This HEMT-PA-based rectifier is 63.94% efficient. Once again, Z_{load} has been reach thanks to the active loop depicted on Fig. 2. The measured RF power sweep is shown in Fig. 7. As shown with Circuit-A, we can notice the RF power located

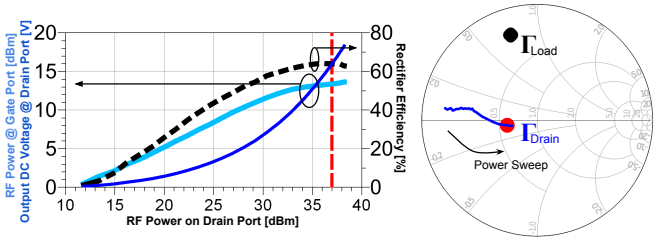

Fig. 7. Circuit-B best rectifying performances with $VGG = -4.7V$, $RD = 80\,\Omega$ and a RF impedance load applied on the gate port $Z_{load} = (9.8 + j\,35.75)\,\Omega$. The circuit exhibit a rectifying efficiency $\eta_R = 63.94\%$ with 37dBm RF power injected on the drain port. RF power at the gate port (light blue), output DC voltage (dark blue) and rectifying efficiency (black dash) are display on the left chart. Right chart display the rectifier's input impedance Γ_{Drain} (drain port) and load impedance Γ_{Load} (gate port).

at the gate port (light blue line) making possible the use of a passive impedance to reach a good rectifying efficiency.

Finally, table II summaries performances of the two X-Band GaN MMICs for both amplifier and rectifier operation modes. In the case of amplifier mode, η_{PA} is the displayed *efficiency*. The *RF power* is the power located at the drain RF port. By presenting good efficiency even at X-Band, this work highlights the possibility to use HEMT GaN for designing a rectifying element. Therefore, there is a need for nonlinear models working both for amplification and rectification. Intrinsic nonlinear capacitances, for example, may be extracted from first and third quadrants.

TABLE II
PERFORMANCES AS AMPLIFIER AND RECTIFIER AT 10.1 GHZ

	Amplifier		Rectifier	
Circuit	A	B	A	B
Max Efficiency(%)	67.87	56.47	64.40	63.94
DC Power (mW)	4186	5112	1671	3182
RF Power (mW)	3281	3362	2594	4976

VIII. CONCLUSION

This work shows measured results on two X-Band GaN MMIC ($0.15\,\mu m$ process). High efficiencies (higher than 56%) are obtained for both amplification and rectification modes. Rectification is performed in self-synchroneous way without any injection on the gate RF port. A passive termination is self-sufficient at microwave frequencies thanks to the Miller effect produced by C_{gd} in the HEMT GaN intrinsic non-linear model.

REFERENCES

[1] *ARM Security Technology: Building a Secure System using TrustZone Technology*, 2009. Ref. no. PRD29-GENC-009492C.

[2] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar. *Innovative instructions and software model for isolated execution. In Hardware and Architectural Support for Security and Privacy C HASP, ACM, page 10*, 2013.

[3] S. Weiser, M. Werner, F. Brasser, M. Malenko, S. Mangard and A. Sadeghi. *TIMBER-V: Tag-Isolated Memory Bringing Fine-grained Enclaves to RISC-V*, Proc. NDSS, 2019.

[4] C. Song, H. Moon, M. Alam, I. Yun, B. Lee, T. Kim, W. Lee, and Y. Paek. *HDFI: Hardware-Assisted Data-Flow Isolation*, In Security and Privacy 16, pages 1C17. IEEE Computer Society, 2016.

[5] A. Waterman and K. Asanovic. *The risc-v instruction set manual, volume i: User-level isa, version 2.2. Technical report, SiFive Inc*, EECS Department, University of California, Berkeley, 2017.